LAB 4: KRACK Attacks

Group 14: *Federico Cucino, Giacomo Checchini, Johnson Katungye*
Network Security Master Course — A.Y. 2022/2023

May 10, 2023

Contents

1 Introduction

1.1 The KRACK Attacks

KRACK, an acronym of **K**ey **R**einstallation **A**tta**c**ks, is a set of vulnerabilities that were discovered in 2017 by Mathy Vanhoef.

This family of attacks exploits the vulnerability found in the 802.11i amendment which is the protocol whose implementation is certified under the trademark WPA and WPA2.
This attack is completed by manipulating the messages of different types of handshakes. Handshakes are mutual authentication protocols made for establishing a temporary key for encrypting the communication between the client and the AP.

Here are listed the handshakes that are reported to be vulnerable to KRACK:

- 4-way handshake
- Fast BSS Transtion handshake
- Group Key handshake
- PeerKey handshake

These attacks trick the victim to reinstall an already-in-use key and they let the adversary replay, decrypt and possibly forge frames during the WiFi communication.

The impact depends on the type of handshake targeted and data confidentiality protocol being used like AES-CCMP, WPA-TKIP and GCMP.
It was also discovered that this attack is particularly devastating against devices running Android version 6.0; in fact, the key is not reinstalled but replaced with one consisting of only 0s.

In our laboratory we focused on the attacks to the 4-way handshake and Fast BSS Transition handshake.

1.2 Attacking the 4-way handshake

In KRACK Attacks, the attacker aims to force the victim into reinstalling an already-in-use key in order to break the security of the data confidentiality and integrity protocol.
The crucial part is that when the victim re-installs the PTK key (the new value is the same of the last key), it triggers the resets of the incremental transmit packet number (called IV in the AES-CCMP protocol) and the replay counter.
All the data confidentiality protocols of the 802.11i amendment behave like stream ciphers. Figure 1 is a scheme which simplifies the encryption mechanism.

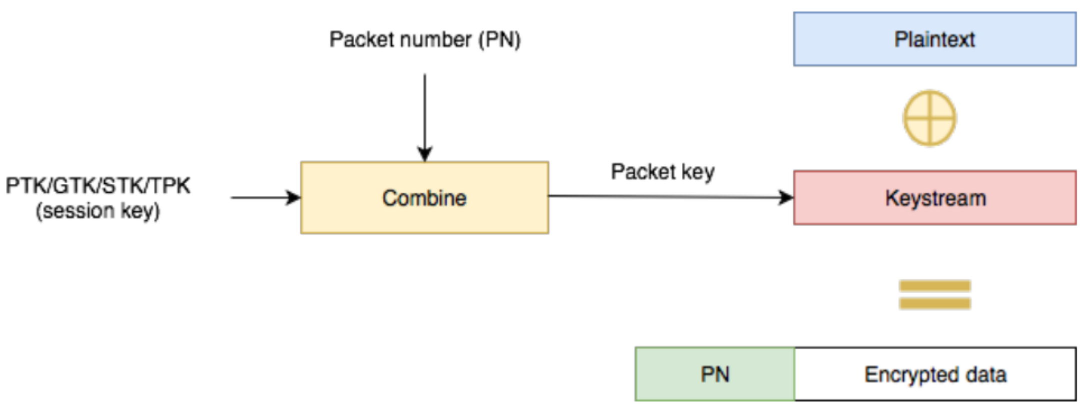

Figure 1: Simplified behaviour of the data confidentiality protocol encryption

As can be seen from Figure 1, encrypted data depends only on three variables: plaintext, packet number and the session key (PTK).
Consequently, if the victim reinstalls a session key that is already in use and resets the packet number, the

same combination will be employed twice generating the same keystream. This is in contrast with the security assumptions of data confidentiality and integrity protocols explicitly outlined in the 2016 standard revision.[1]
We can observe that if a message contains known content using the same keystream, it becomes effortless to derive the used keystream and so to break the security of the data confidentiality protocol.
However, decryption of packets becomes more challenging when most of the content is unknown.

Now that the reasons why reinstalls are detrimental to communication security are understood, it is necessary to understand how the attack can be carried out.
The reason why it is possible is because of a vulnerability found in the handshake definition in the standard itself.
When the supplicant (client) receives retransmitted message 3 of the handshake, it reinstalls the key already installed, resetting the packet number and the replay counter. This behaviour is made on purpose in the standard because messages may be lost or dropped. The AP will retransmit message 3 if it did not receive an appropriate response as acknowledgment. As a result, the client may receive message 3 multiple times. Each time it receives this message, it will reinstall the same session key and resetting the packet number.

Therefore it's possible to attack the 4-way handshake by following these steps:
First of all there is the need to establish a MitM position between AP and client; this is not so easy because you need to clone the real AP into a different channel. Now the MitM only captures and forwards the first three messages and stores these messages to be able to use them later in the communication. After that, the attacker blocks the transmission of message4 but the client starts to transmit data believing that the handshake is completed. The first packet number of the data will be 1. The AP thinks that the message was lost so it retransmits message3.
Because the PTK has been previously installed, the victim answers with the encrypted message4 in response and it re-installs the same PTK. The first packet number of this data will have again value 1. The complete process is showed in Figure 2.

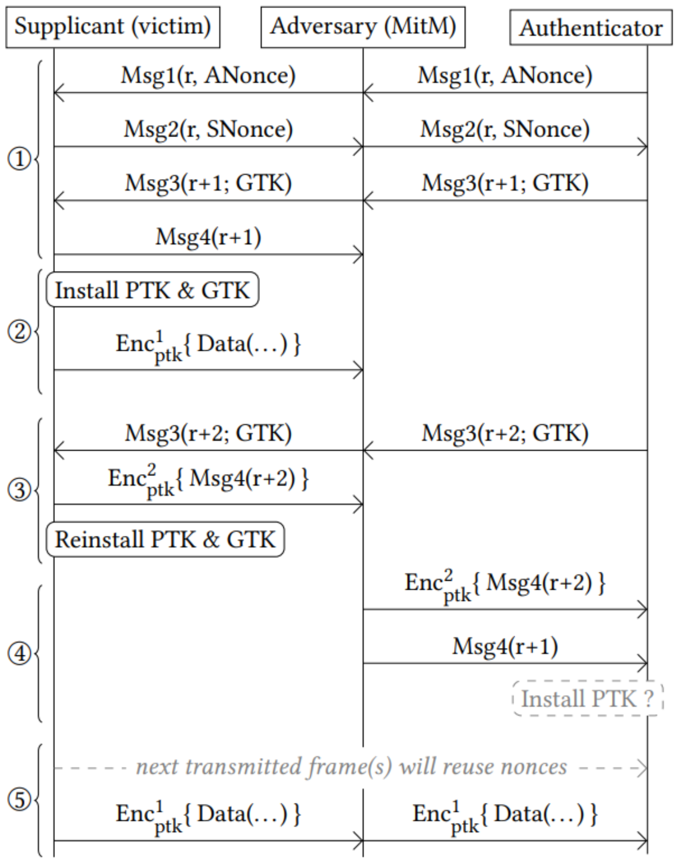

Figure 2: Attacking the 4-way handshake, from "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2", Mathy Vanhoef, 2017

[1]IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements, Part 11. IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)

1.2.1 Simulation script

To better consolidate theory concepts seen during the lab, we created a Python script in which are represented the main steps of the KRACK attack.

In particular, it's represented the attack to a 4-way handshake protocol in which the client accepts the plaintext retransmission of message 3 of the handshake.

This is only a simple simulation and the code can be viewed on the Github repository[2] in the folder called `simulation`.

To run it you only need to have Python3 installed in your machine. In the Virtual Machine of our laboratory the script are already downloaded and you have only to enter in the right folder in order to run them. Open the terminal and follow this commands in order to start them correctly.

```
~$ cd netsec-krack-attack
~/netsec-krack-attack$ cd simulation
~/netsec-krack-attack$ ./start.sh
```

Once the three terminals appear, we suggest to place them as it's shown in Figure 3 in order to better follow the flow of the attack. The last step is to start the simulation in the *Authenticator/AP terminal* by pressing `<enter>`. To continue you have to press `<enter>` in the MitM terminal until the program is terminated.

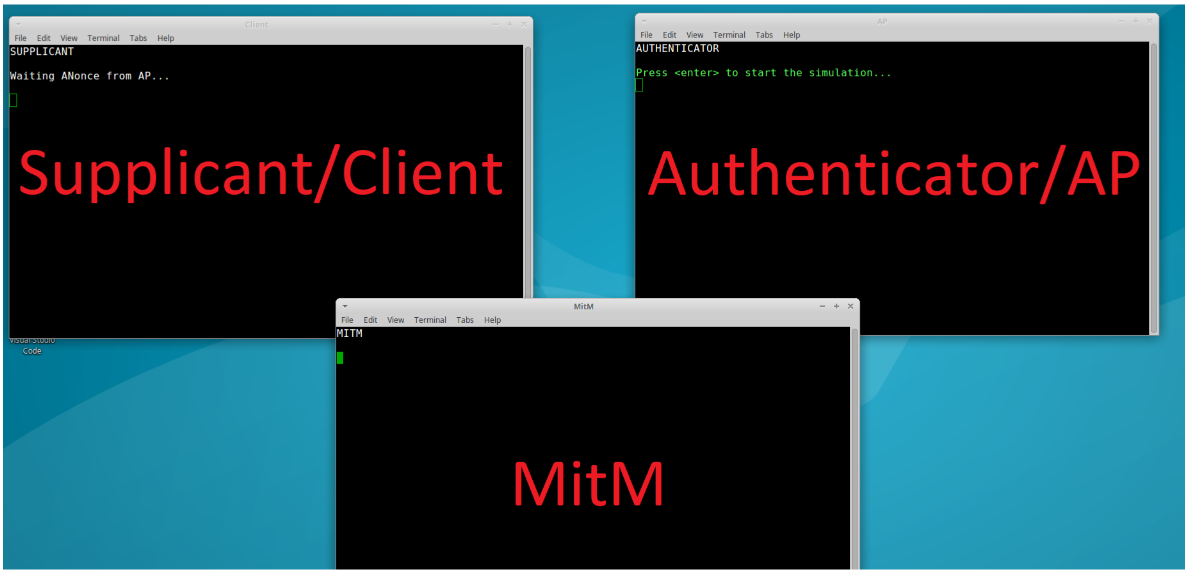

Figure 3: Screenshot from the VM used in the laboratory

1.3 Attacking the Fast BSS Transition (FT) handshake

The Fast BSS Transition handshake is another handshake protocol that is vulnerable to KRACK Attacks.

It reduces the roaming time when a client moves from one AP to another one of the same protected network. This handshake is similar to the 4-way handshake but the direction of the messages is inverted: the client starts the handshake.

The attack at the FT handshake aims to reinstall the key in the AP instead of the client. This handshake is theoretically secure but, as discovered by Vanhoef through practical tests and code inspections, it has a real vulnerability in its implementations.

In this case, the PTK is reinstalled after receiving a reassociation request (the third message of the FT handshake) as was for the message3 of the 4-way handshake.

To exploit this vulnerability we do not require a MitM position but only the ability to eavesdrop and injecting frames is sufficient in order to complete the attack.

In the attack itself we let client and AP complete a FT handshake and exchange some encrypted packets. After that we replayed the reassociation request to the AP. It will proceed to reinstall the PTK and start again the

[2]Repo: `https://github.com/turbostar190/netsec-krack-attack`

numeration of the packets, breaking the security of the communication. In Figure 4 there is showed the entire process.

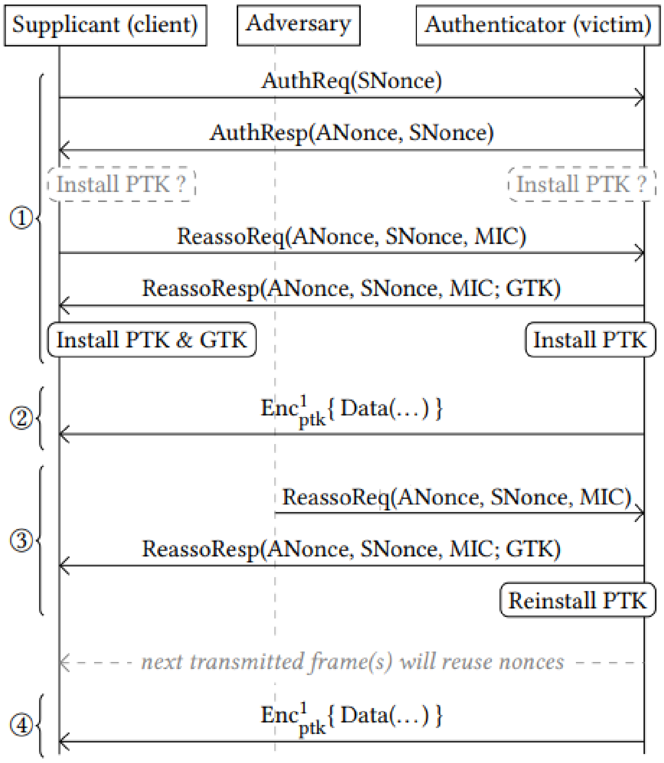

Figure 4: Attacking the Fast BSS Transition handshake, from "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2", Mathy Vanhoef, 2017

2 Simulation setup

This section explain how to build the required environment.

2.1 Building the tools

The first recommended step is to create a virtual machine running (X)Ubuntu 16.04 x64. Set up and do not install any available updates.

Any following command is now meant to be run inside the Virtual Machine!

First of all we have to downgrade the kernel, because some APIs have been fixed and have been backported for some version. You can check your current kernel version typing `uname -r`.

We will install kernel 4.4.0-040400 of January 2016 from this page: download *linux headers*, *linux headers generic* and *linux image generic* in a folder. From that directory, install them doing `sudo dpkg -i linux*.deb` Reboot, hold *shift* to show Grub menù and then select "Advanced options" >"4.4.0".

Double check your current version again with command `uname -r`. Should now be *4.4.0*.

Open a terminal to clone and setup the `netsec-krack-attack`[3] repository and its dependencies. This repo contains all the needed to run our tests.

```
~$ sudo apt update
~$ sudo apt install git libnl-3-dev libnl-genl-3-dev pkg-config \
     libssl-dev net-tools git sysfsutils virtualenv wireshark
~$ git clone --recurse-submodules https://github.com/turbostar190/netsec-krack-attack
```

[3]Repo: `https://github.com/turbostar190/netsec-krack-attack`

```
~$ cd netsec-krack-attack/
```

At this point, we have to build the required packages from the source and to create the Python3 virtual environment needed to execute the analysis scripts later.

Let's start with the `mininet-wifi` folder:

```
~/netsec-krack-attack$ cd mininet-wifi
~/netsec-krack-attack/mininet-wifi$ sudo util/install.sh -Wlnfv
```

Let's switch to the `krackattacks-script` folder and let's run its suite install script:

```
~/netsec-krack-attack/mininet-wifi$ cd ../krackattacks-scripts
~/netsec-krack-attack/krackattacks-scripts$ cp wpa_supplicant/patchedconfig \
                                              wpa_supplicant/.config
~/netsec-krack-attack/krackattacks-scripts$ cp wpa_supplicant/patchedconfig \
                                              hostap-wpa_supplicant-2.3/wpa_supplicant/.config
~/netsec-krack-attack/krackattacks-scripts$ cp wpa_supplicant/patchedconfig \
                                              hostap-wpa_supplicant-2.5/wpa_supplicant/.config
~/netsec-krack-attack/krackattacks-scripts$ cd krackattack
~/netsec-krack-attack/krackattacks-scripts/krackattack$ ./build.sh
~/netsec-krack-attack/krackattacks-scripts/krackattack$ ./pysetup.sh
~/netsec-krack-attack/krackattacks-scripts/krackattack$ sudo ./disable-hwcrypto.sh
~/netsec-krack-attack/krackattacks-scripts/krackattack sudo reboot
```

After a system reboot, we have the full environment to run our tests.

2.2 Repo Structure

The root of the repo contains two files `krack-topology-ft.py` and `krack-topology-client.py`. These are the two different topologies we will use to test CVE-2017-13082 vulnerability and all the others CVE-2017-130{77-81} respectively.
Subfolders contains the analysis scripts themselves, as said before. These scripts are launched directly by the Mininet-wifi topology wizard.

```
.
|-- dump
|   |-- ...
|-- krackattacks-scripts
|   |-- hostap-wpa_supplicant-2.3
|   |   |-- wpa_supplicant
|   |   |   |-- (.config)
|   |   |   |-- (patchedconfig)
|   |-- hostap-wpa_supplicant-2.5
|   |   |-- wpa_supplicant
|   |   |   |-- (.config)
|   |   |   |-- (patchedconfig)
|   |-- krackattack
|   |   |-- krack-ft-test.py
|   |   |-- krack-test-client.py
|   |   |-- hostapd.conf
|   |-- wpa_supplicant
|   |   |-- (.config)
|   |   |-- patchedconfig
|   |-- ...
|-- mininet-wifi
|   |-- ...
|-- mininet-wifi
|   |-- ap.py
|   |-- client.py
|   |-- mitm.py
```

```
|   |-- start.sh
|-- client-network.conf
|-- krack-topology-client.png
|-- krack-topology-client.py
|-- krack-topology-ft.png
|-- krack-topology-ft.py
|-- README.md
```

3 Testing hands-on

This section explains how to test the different vulnerabilities inside and outside Mininet.

3.1 Testing Mininet vulnerable access points

To test the CVE-2017-13082 Fast Transition protocol vulnerability, we have to first associate to an access point and then the client have to ask to roam to another access point in its range.

The topology made by Mininet-wifi is shown in Table 1 and Figure 5.

Table 1: Topology

Device	Code	MAC Address	IP Address
Station 1	sta1	*random*	10.0.0.1/8
Access Point 1	ap1	02:00:00:00:01:00	10.0.0.101/8
Access Point 2	ap2	02:00:00:00:02:00	10.0.0.102/8

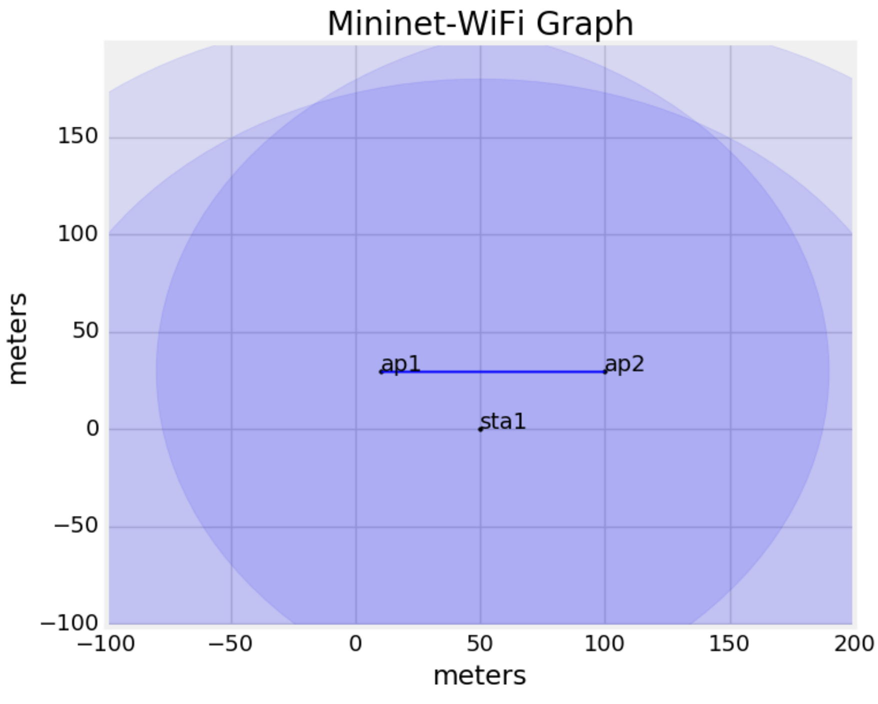

Figure 5: FT topology

This topology contains three devices: station (sta1), access point 1 (ap1) and access point 2 (ap2).

From the root of the repo folder, run `sudo ./krack-topology-ft.py` in a terminal.

The main terminal manages Mininet-wifi, while spawned XTerms control direcly a device: the "KrackAttack: sta1" XTerm is the script listening to a roaming reassociation request on the client station and the "wpa_cli: sta1" XTerm is the terminal we will use to ask the device to roam. (See Figure 6)

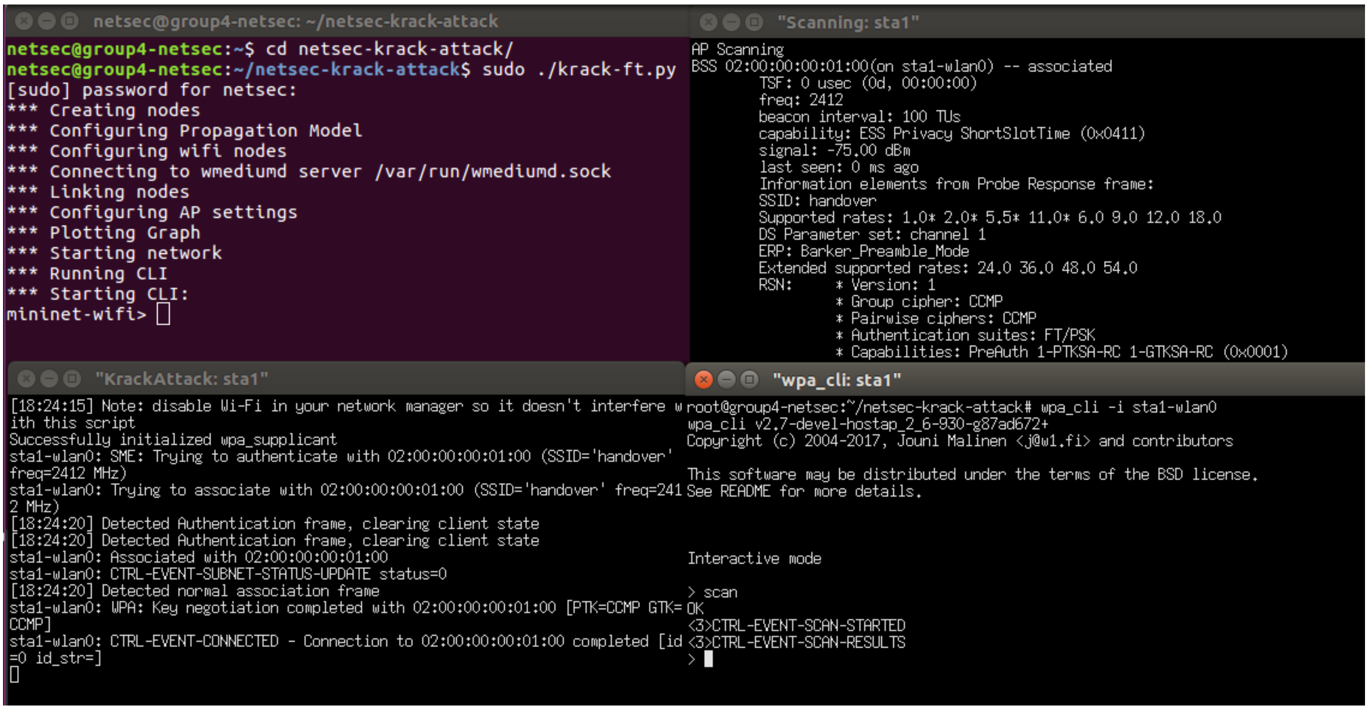

Figure 6: FT simulation startup

We can test the correct behaviour of the *packet number aka Initialization Vector (IV)* value by generating unicast traffic with `sta1 arping -I sta1-wlan0 -c 10 10.0.0.101` in the main terminal.
We can see in the `KrackAttack` term that every packet has its own increasing IV and sequence number. (See Figure 7)

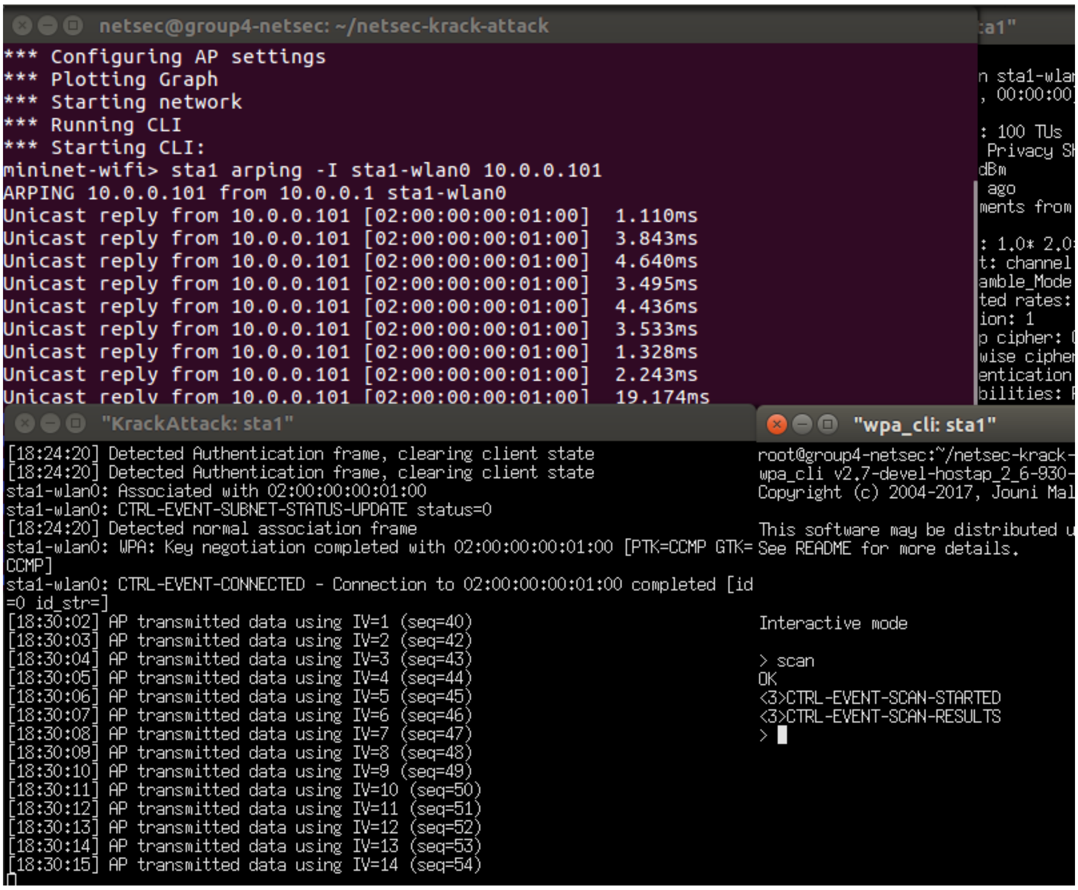

Figure 7: FT simulation AP1 ping

Let's ask the roaming: inside the "wpa_cli" terminal, type `wpa_cli -i sta1-wlan0`.
Then, type `scan`, and wait a few seconds for both `<3>CTRL-EVENT-SCAN-STARTED` and `<3>CTRL-EVENT-SCAN-RESULTS` signals to appear on the screen.
Typing `scan_results` we should see the two different access points.
Ask the client to do the transition to the other AP with `roam 02:00:00:00:02:00`.

The listener script should now detect the *Reassociation* frame and starts replaying it. Generating unicast traffic to this access point from the main terminal, using the command `sta1 arping -I sta1-wlan0 -c 10 10.0.0.102`, we can notice that the Initialization Vector values are reset at every malicious replay. (See Figure 8)

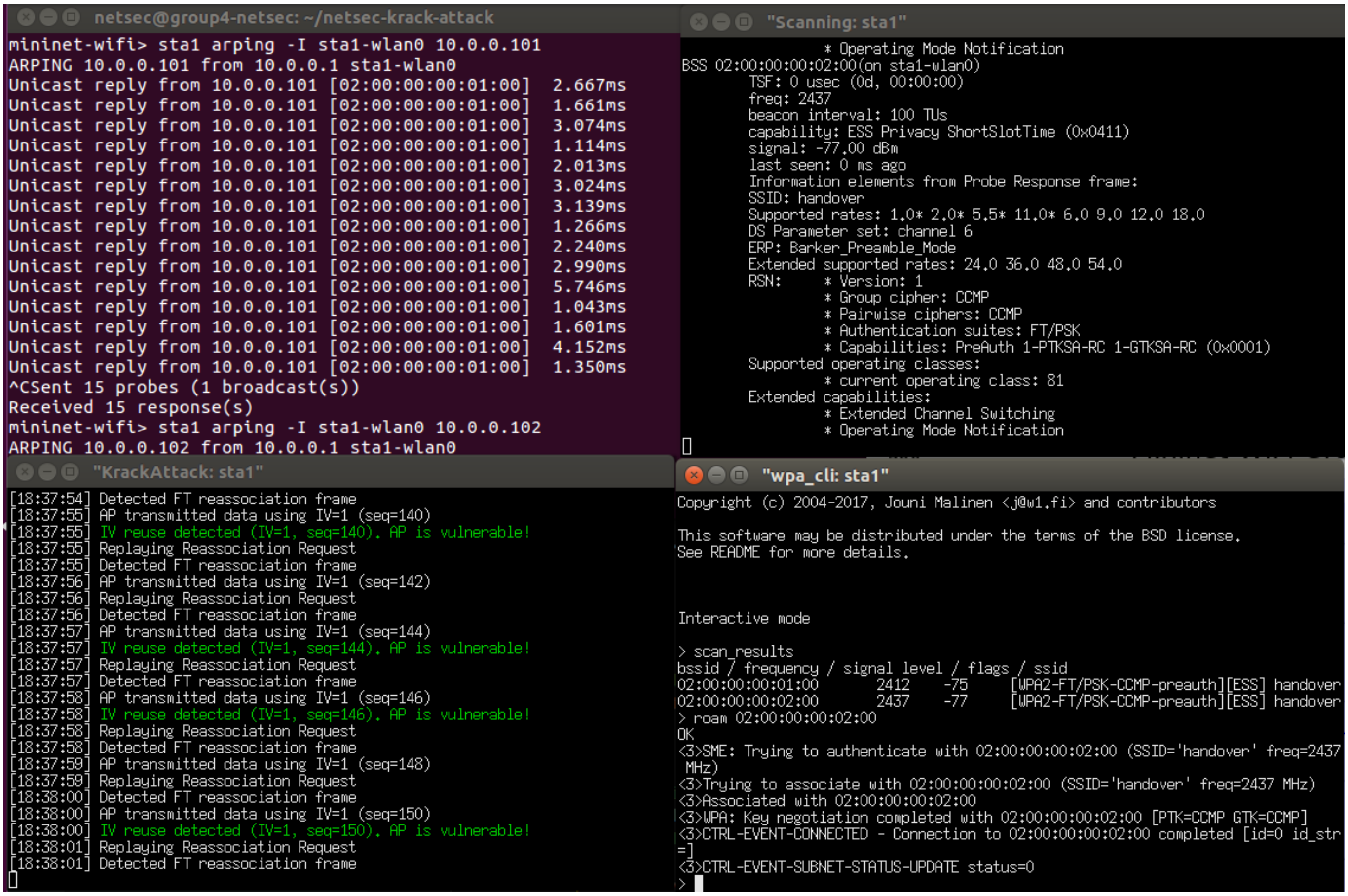

Figure 8: FT simulation AP2 ping after roaming

To stop the simulation, *ctrl + d* in the main terminal and close every opened Xterm.
Finally, execute `sudo mn -c` to perform a complete cleanup of Mininet things and routes.

Mission complete!

3.2 Testing a Mininet non-vulnerable client

This section tests for key reinstallations in the 4-way handshake by repeatedly sending encrypted message 3's to the client. In other words, this tests for CVE-2017-13077 (the vulnerability with the highest impact) and for CVE-2017-13078.

The topology we will adopt on Mininet contains two devices: station (sta1) and access point 1 (ap1) and it's described as in Table 2 and Figure 9.

Table 2: Topology

Device	Code	MAC Address	IP Address
Station 1	sta1	*random*	192.168.100.100/24
Access Point 1	ap1	02:00:00:00:01:00	192.168.100.254/24

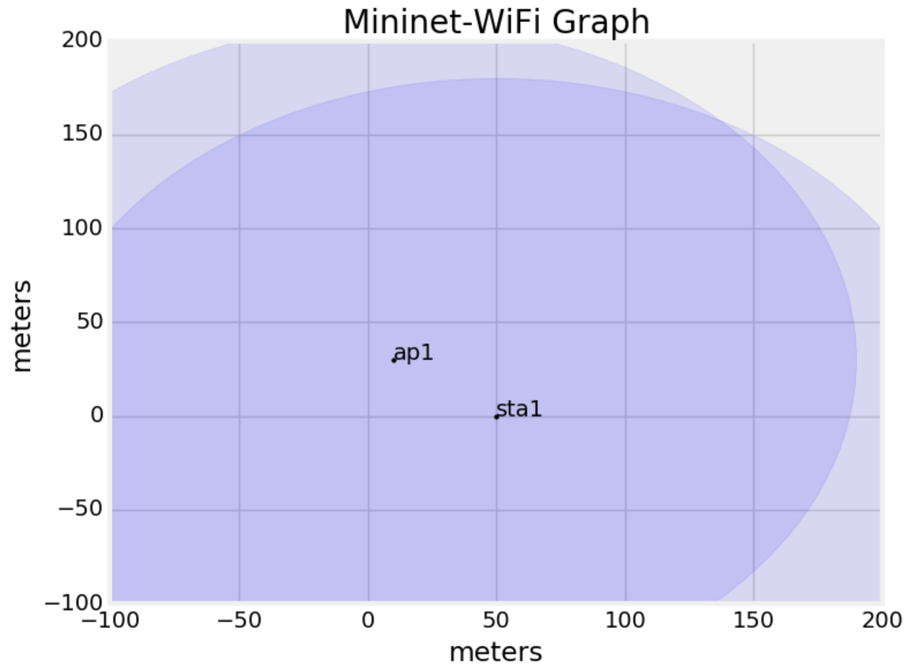

Figure 9: Topology

The script monitors traffic sent by the client to see if the pairwise key is being reinstalled. Note that this effectively performs two tests: whether the pairwise key is reinstalled, and whether the group key is reinstalled.

Before starting, we must update the file
∼`/netsec-krack-attack/krackattacks-scripts/krackattack/hostapd.conf`
to change the line starting with *interface=wlan0* into *interface=ap1-wlan0.*

From the root of the repo folder, run `sudo ./krack-topology-client.py` in a terminal (twice, because the first time it will always throw an error).

There are three different terminals: main one, "AP: ap1" xterm and "Connection: sta1" xterm.
Without doing anything else, we should see the "Connection" term on Station 1 trying to connect to the (virtual) wifi network generated by the AP term (see Figure 10). The default credentials are:

- SSID: testnetwork
- Psk: abcdefgh

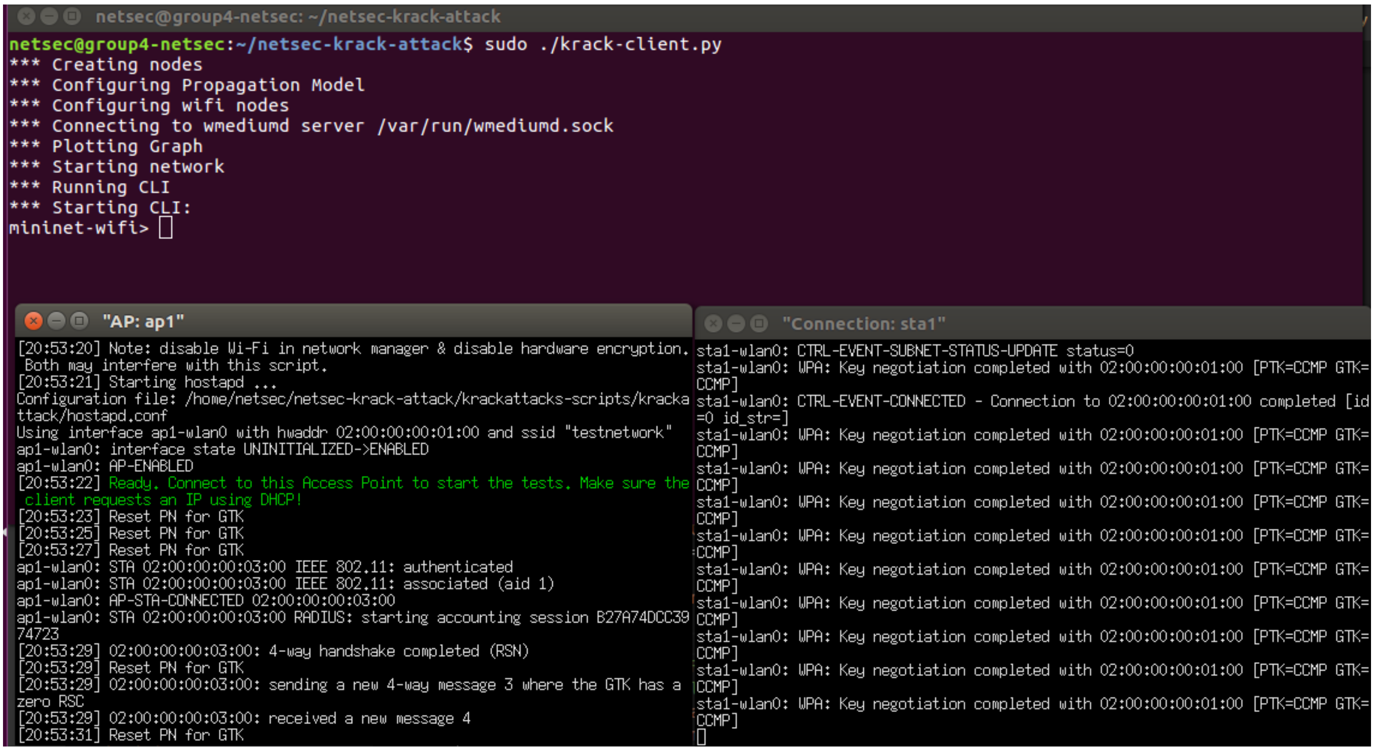

Figure 10: Client script

In the AP term we should see the script resetting the Packet Number and sending a new 4-way message 3. This specific environment is not vulnerable, and we can read that the station replies with a new message 4 every time without the same Initialization Vector.

To run this test in a vulnerable environment please refer to Section 3.3.

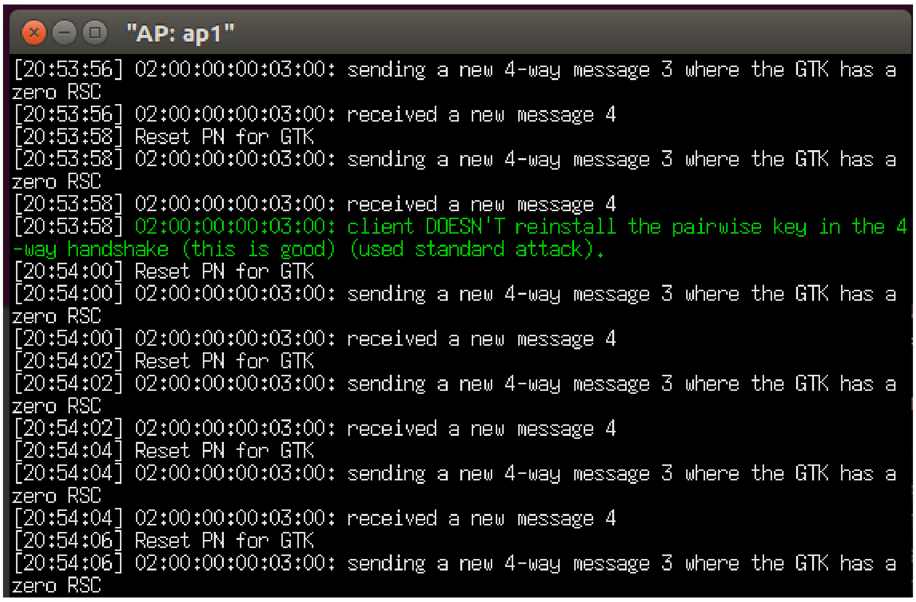

Figure 11: Not vulnerable client

Finally, execute `sudo mn -c` to perform a complete cleanup of Mininet things and routes.

3.3 Testing a Mininet vulnerable client

To test a vulnerable station on Mininet we have to downgrade `wpa_supplicant`, the software used to manage the connection and the communication between a wireless access point and the kernel. The installed version by the setup done earlier is v2.7, the first patched one against 4-way handshake message 3 replay.

3.3.1 Downgrade the kernel

Version 2.3, available inside *hostap-wpa_supplicant-2.3* directory, is vulnerable to *IV reuse*, reinstalling the pairwise key.
To install it, just do what follows:

```
~/netsec-krack-attack$ cd krackattacks-scripts/hostap-wpa_supplicant-2.3/wpa_supplicant/
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.3/wpa_supplicant$ make clean
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.3/wpa_supplicant$ make
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.3/wpa_supplicant$ sudo make instal
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.3/wpa_supplicant$ cd ../../../
```

After this, if we do the same steps as described in Section 3.2 we should achieve a yellow warning stating the reuse of the Initialization Vector, like show in Figure 12.

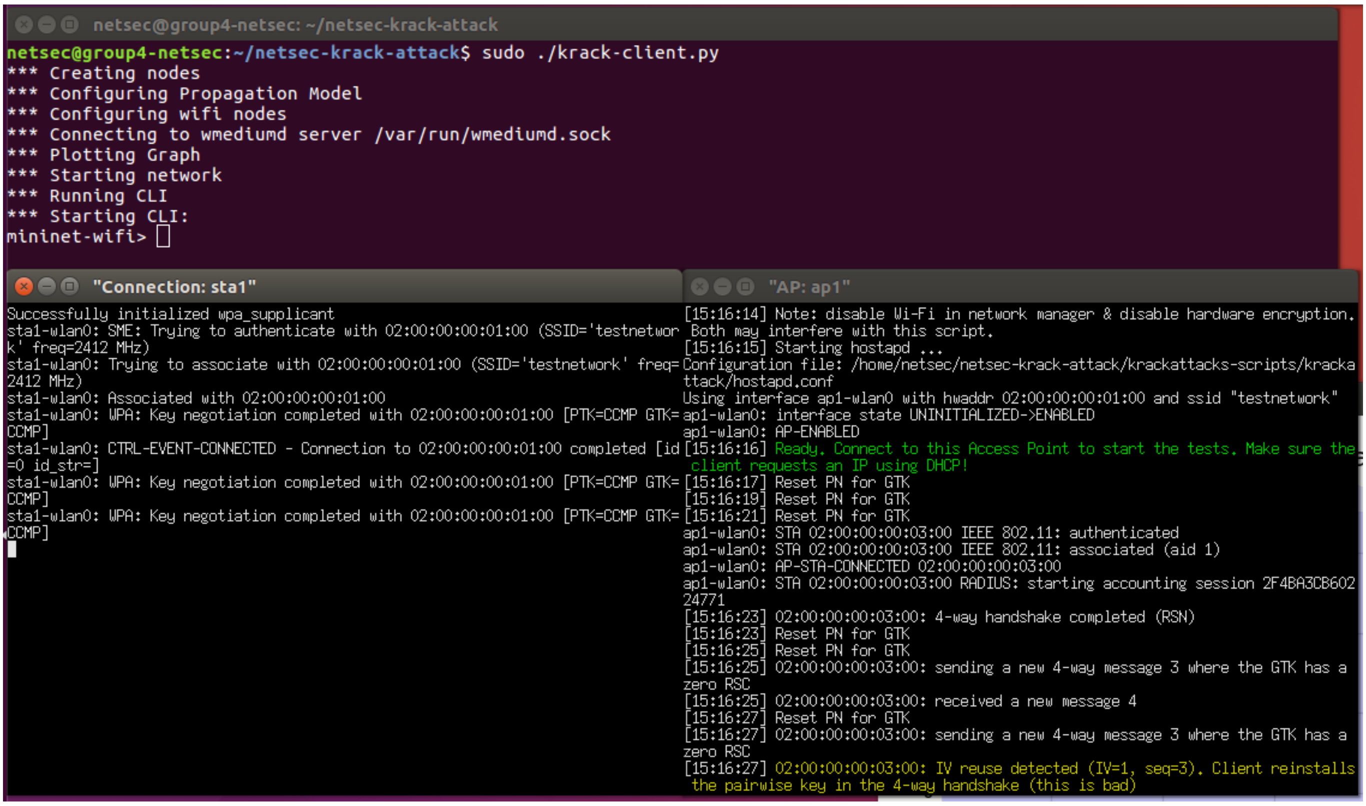

Figure 12: Vulnerable client, *wpa_supplicant v2.3*

A even more vulnerable package version is *v2.5*. As before, just change to the appropriate folder and repeat the installation of the package.

```
~/netsec-krack-attack$ cd krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant/
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant$ make clean
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant$ make
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant$ sudo make instal
~/netsec-krack-attack/krackattacks-scripts/hostap-wpa_supplicant-2.5/wpa_supplicant$ cd ../../../
```

There is not only the reuse of the Initialization Vector, but furthermore also the usage of an all-zero key. See Figure 13.

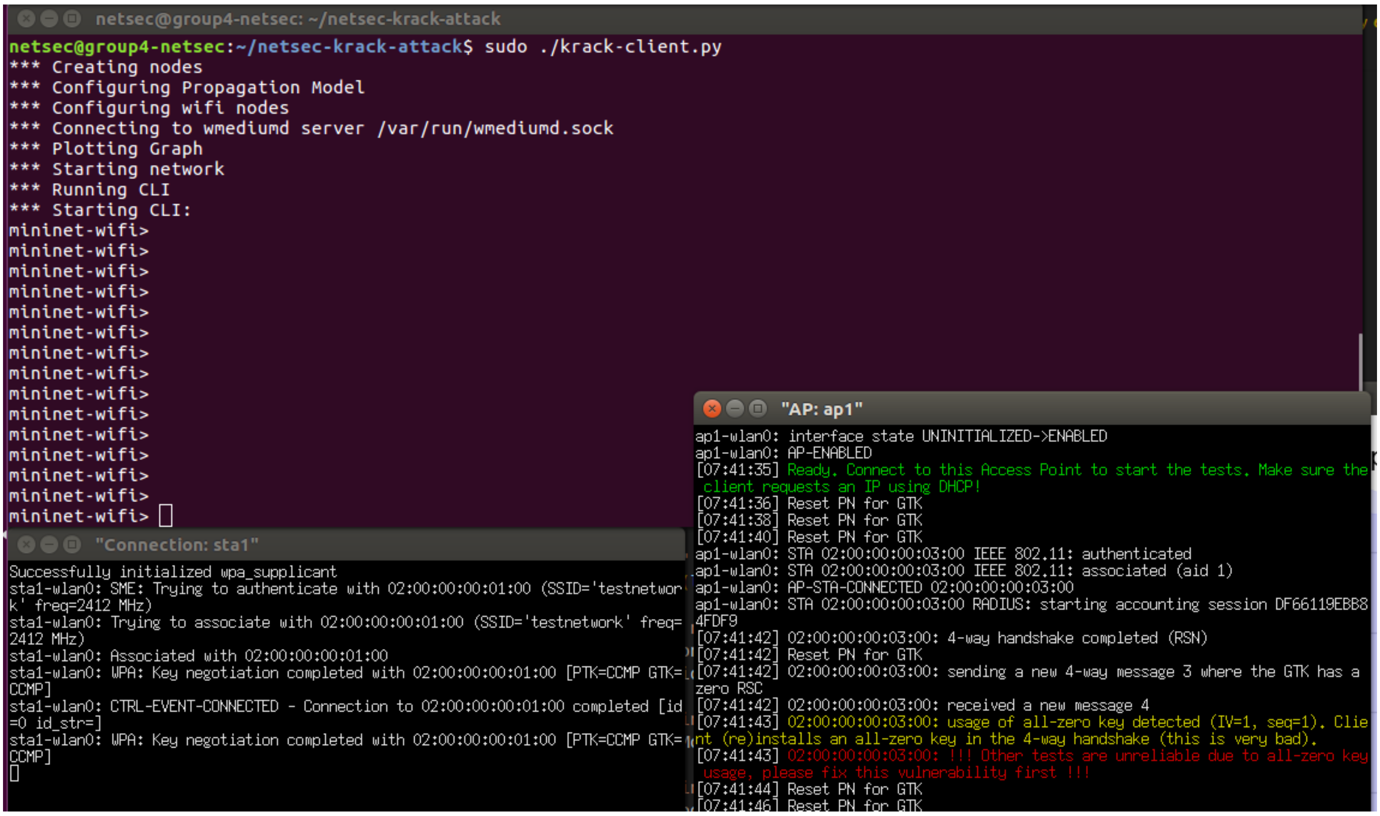

Figure 13: Vulnerable client, *wpa_supplicant v2.5*

3.4 Testing a real vulnerable client

To test a vulnerable environment, we setup a real test using an external wifi dongle and an Android 6 device.

Differently from the previuos simulation, we will obviously skip the Mininet-wifi setup and we will jump straight to the analysis script made by Mathy Vanhoef, the author of the paper describing these vulnerabilities.

Before starting, we have to turn off the wifi from our connection manager and we have to get the interface name of our wifi dongle. One way could be with the command `iw dev`. For example, our dongle interface name is `wlxa0f3c1085c6a`.

At this point, we have to update the file ~`/netsec-krack-attack/krackattacks-scripts/krackattack/hostapd.conf` to change the line starting with *interface=ap1-wlan0* into *interface=<interface-name>*, where *<interface-name>* is our wifi dongle interface.

Let's navigate to `krackattacks-scripts/krackattack` folder.

First we have to enter sudo mode with `sudo su` and access the Python3 virtual environment typing `source venv/bin/activate`. We can now type `python ./krack-test-client.py`.

We should see from the phone the newly generated network, just connect to it and we can verify the issues with the protocol and the reinstallation of the key.

```
(venv) root@group4-netsec:/home/netsec/netsec-krack-attack/krackattacks-scripts/krackattack#
./krack-test-client.py
[21:12:38] Note: disable Wi-Fi in network manager & disable hardware encryption. Both may int
erfere with this script.
[21:12:39] Starting hostapd ...
Configuration file: /home/netsec/netsec-krack-attack/krackattacks-scripts/krackattack/hostapd
.conf
Using interface wlxa0f3c1085c6a with hwaddr a0:f3:c1:08:5c:6a and ssid "testnetwork"
wlxa0f3c1085c6a: interface state UNINITIALIZED->ENABLED
wlxa0f3c1085c6a: AP-ENABLED
[21:12:40] Ready. Connect to this Access Point to start the tests. Make sure the client reque
sts an IP using DHCP!
[21:12:41] Reset PN for GTK
[21:12:43] Reset PN for GTK
[21:12:45] Reset PN for GTK
wlxa0f3c1085c6a: STA d0:87:e2:89:ef:4f IEEE 802.11: authenticated
wlxa0f3c1085c6a: STA d0:87:e2:89:ef:4f IEEE 802.11: associated (aid 1)
wlxa0f3c1085c6a: AP-STA-CONNECTED d0:87:e2:89:ef:4f
wlxa0f3c1085c6a: STA d0:87:e2:89:ef:4f RADIUS: starting accounting session 22A81D1139E807D4
[21:12:46] d0:87:e2:89:ef:4f: 4-way handshake completed (RSN)
[21:12:47] Reset PN for GTK
[21:12:47] d0:87:e2:89:ef:4f: sending a new 4-way message 3 where the GTK has a zero RSC
[21:12:47] d0:87:e2:89:ef:4f: received a new message 4
[21:12:48] d0:87:e2:89:ef:4f: DHCP reply 192.168.100.2 to d0:87:e2:89:ef:4f
[21:12:49] Reset PN for GTK
[21:12:49] d0:87:e2:89:ef:4f: sending a new 4-way message 3 where the GTK has a zero RSC
[21:12:50] d0:87:e2:89:ef:4f: client has IP address -> now sending replayed broadcast ARP pac
kets
[21:12:50] d0:87:e2:89:ef:4f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent
 0 ARPs this interval)
[21:12:50] d0:87:e2:89:ef:4f: IV reuse detected (IV=1, seq=3). Client reinstalls the pairwise
 key in the 4-way handshake (this is bad)
[21:12:51] Reset PN for GTK
[21:12:52] d0:87:e2:89:ef:4f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent
 1 ARPs this interval)
```

Figure 14: Vulnerable client, Android 6

If we try with a more recent device, we should see the same behaviour as the Section 3.2.

4 Detecting the attack in Wireshark

Wireshark is launched in every Mininet simulation exercises and it will intercepts all the generated traffic.
Once Wireshark is opened, you need to capture the *mon0* interface. When the simulation is finished, it is possible to analyze all the captured packets to be able to detect whether the KRACK attack was successful or not.
To do it, we only need to apply the correct filters like, for example, selecting the MAC address of the source (`wlan.sa == 02:00:00:00:01:00`) to analyze the Fast Transition dump.

wlan.fc.subtype != 0x8 && wlan.sa == 02:00:00:00:01:00

No.	Time	Source	Destination	Protocol	Length	CCMP Ext. Initialization Vector	Info
8	0.451674198	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=166, FN=0, Flags=........
10	0.471317077	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=167, FN=0, Flags=.p....F.
22	1.495576347	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000002	Data, SN=168, FN=0, Flags=.p....F.
25	1.504711431	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=169, FN=0, Flags=........
38	2.522846232	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=170, FN=0, Flags=.p....F.
40	2.526505245	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=171, FN=0, Flags=........
52	3.547398228	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=172, FN=0, Flags=.p....F.
55	3.552807459	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=173, FN=0, Flags=........
67	4.570142510	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=174, FN=0, Flags=.p....F.
70	4.578626817	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=175, FN=0, Flags=........
82	5.590638860	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=176, FN=0, Flags=.p....F.
85	5.597897815	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=177, FN=0, Flags=........
97	6.614063056	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=178, FN=0, Flags=.p....F.
100	6.621900848	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=179, FN=0, Flags=........
112	7.639818671	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=180, FN=0, Flags=.p....F.
115	7.652676981	02:00:00:00:01:00	02:00:00:00:00:00	802.11	265		Reassociation Response, SN=181, FN=0, Flags=........
127	8.666967969	02:00:00:00:01:00	02:00:00:00:00:00	802.11	150	0x000000000001	Data, SN=182, FN=0, Flags=.p....F.

Figure 15: Traffic captured in the Fast BSS Transtion handshake excercise

As you can see from the Figure 15, the result that the attack was successful is very clear because after every Reassociation Response received by the client there is a data packet in which the packet number it's always equal to 1.

In the Github repository there is also a folder called `dump` in which you can find precaptured packets for every exercises seen in this laboratory. To open it in Wireshark it's only necessary to click File >> Open and select the desidered file in the `dump` folder.